Remarks/Arguments

The preceding amendments and following remarks are submitted in response to the Office Action of the Examiner mailed August 16, 2004, setting a three month shortened statutory period for response ending November 16, 2004. Claims 1-8 and 10-24 remain pending, with claims 13-24 being newly presented. Claim 9 has been canceled without prejudice. Reconsideration, examination and allowance of all pending claims are respectfully requested.

In paragraph 3 of the Office Action, the Examiner rejected claims 7-9 under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. The Examiner states that the claimed subject matter as a whole must accomplish a practical application (i.e. must produce a "useful, concrete and tangible result").

Claim 7, as amended, recites:

> 7. (Currently Amended) An object adapted for persistent storage, the object [Emphasis Added] having a smart pointer, wherein the smart pointer includes an address attribute for containing the address of an object, and an object unique identifier attribute for containing the unique identifier of an object, wherein the object smart pointer has an assignment operation which stores the address of the object being pointed to and the unique identifier of the object being pointed to, and wherein the object includes a load method for using the smart pointer unique identifier attribute to determine and load a new smart pointer address attribute after the object being pointed to is loaded from persistent storage [Emphasis Added].

As can be seen, claim 7 now recites an object adapted for persistent storage, wherein the object includes a load method for using the smart pointer unique identifier attribute to determine and load a new smart pointer address attribute after the object being pointed to is loaded from persistent storage [Emphasis Added]. The claim now clearly recites an object that is useful, concrete and produces a tangible result. More specifically, claim 7 recites an object that is

loaded into memory from persistent storage that has an updated smart pointer address attribute after the object that is being pointed to is loaded from persistent storage. Clearly, such an object is useful, and produces a tangible result. For these and other reasons, claim 7 and dependent claim 8 are believed to fully comply with 35 U.S.C. § 101. Claim 9 has been canceled without prejudice.

In paragraph 5 of the Office Action, the Examiner rejected claims 1-10 under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,125,364 to Greef et al. After careful review, Applicant must respectfully disagree.

Claim 1 recites:

> 1. (Previously Presented) A method for creating a plurality of objects from data in persistent storage, the objects having object pointers, unique object identifiers, and object types as attributes, the method comprising the steps of:
>
> reading the total number of type sets;
>
> for each type set, reading the total number of objects in each type set;
>
> for each object in said type set, creating an object from said data in persistent storage;
>
> for each object pointer in said objects, obtaining the unique object identifier corresponding to said object pointer; and
>
> for each obtained unique object identifier, obtaining the object address corresponding to said unique object identifier and setting each of said object pointers to said corresponding object addresses.

As can be seen, claim 1 recites, among other things, the steps of: (1) obtaining the unique object identifier corresponding to said object pointer for each object pointer in said objects; and (2) obtaining the object address corresponding to said unique object identifier and setting each of said object pointers to said corresponding object addresses for each obtained unique object identifier. Greef et al. does not appear to disclose or suggest these steps.

The Examiner cites to column 4, lines 59-60 for anticipating the step labeled (1) above.

Column 4, lines 52-61 of Greef et al. states:

> When an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. This is done by invoking an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier. The Data Store returns a Data Cursor with the object's contents. <u>A smart pointer is then created and the smart pointer asks the Class that corresponds to the object's class identifier, to create an object of that class</u> [Emphasis Added].

As can be seen, this passage of Greef et al. merely states that when loading on object, a smart pointer is created <u>which then asks the Class that corresponds to the object's class identifier to create an object of that class</u> [Emphasis Added]. However, this clearly does not disclose or suggest obtaining a unique object identifier that corresponds to the object pointer <u>for each object pointer in the objects</u>, as recited in claim 1. In fact, the above passage of Greef et al. appears to relate to a method for creating an object during an object load, while the step labeled (1) above relates to a method for resolving object pointers within an object during an object load.

Next, the Examiner cited to column 4, lines 59-61 of Greef et al. as disclosing the step labeled (2) above. The passage at column 5, lines 59-61 of Greef et al. has already been reproduced above. The cited passage of Greef et al. clearly does not disclose obtaining the object address corresponding to the unique object identifier and <u>setting each of said object pointers to said corresponding object addresses</u> [Emphasis Added] for each obtained unique object identifier, as recited in claim 1. As noted above, the cited passage of Greef et al. appears to relate to a method of creating an object during an object load, while the step labeled (2) above relates to a method for resolving object pointers within an object during an object load.

In addition to the foregoing, Greef et al. state:

> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. <u>All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves</u> [Emphasis Added].

(Greef et al., column 4, lines 32-41). As such, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to obtain <u>the object address</u> corresponding to the unique object identifier and <u>setting each of said object pointers to said corresponding object addresses</u> for each obtained unique object identifier, as recited in claim 1. In view of the foregoing, claim 1 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claims 2-3 are also believed to be clearly patentable over Greef et al.

Specifically with respect to claim 2, the Examiner states that Greef et al. disclose objects created in a first pass and that the objects' pointers values are set in a second pass subsequent to the first pass (citing Figure 10, items 404 and 407 of Greef et al., and the corresponding sections of the description). In reviewing Greef et al., item 404 of Figure 10 appears to relate to creating an object of a class, and item 407 appears to relate to initializing the object. Nothing in item 404, item 407 or the corresponding disclosure of Figure 10 of Greef et al. appears to disclose setting each of said object pointers to the corresponding object addresses during a second pass, as recited in claim 2. In fact, and as noted above, Greef et al. state:

> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a

> corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. <u>All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves</u> [Emphasis Added].

(Greef et al., column 4, lines 32-41). As can be seen, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to set "each of said object pointers to said corresponding object addresses during a second pass", as recited in claim 2. For these additional reasons, dependent claim 2 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claim 3 is believed to be clearly patentable over Greef et al.

Turning now to claim 4, which recites:

> 4. (Currently Amended) A method for writing a plurality of objects in non-persistent storage to persistent storage, the objects having pointers to objects, unique object identifiers, and object types as attributes, the method comprising the steps of:
>
> <u>providing one or more common interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage</u> [Emphasis Added];
>
> grouping together said objects into type sets, wherein each of said objects in each of said type sets have the same type, wherein each of said type sets have a set population equal to a total number of objects inhabiting said type set;
>
> counting each of said type sets and arriving at a total number of sets;
>
> converting each of said objects to a persistable form including obtaining a persistable form for each of said pointers to objects by obtaining a unique object identifier corresponding to each of said pointers to objects;
>
> writing said total number of type sets to persistent storage; and
>
> writing each of said type sets to persistent storage.

Greef et al. do not appear to disclose or suggest <u>providing one or more common interfaces that are used by each of the plurality of objects to write the objects from non-persistent storage to persistent storage</u>. In fact, the persistent object classes of Greef et al. do not appear to be interfaced based as all.

In one illustrative embodiments of the present invention, the one or more common interfaces may include and be defined for the IPersistFile 20, IUnknown 28, and IPersistStream 36 classes, which are pre-defined by the COM (Component Object Model) specification (see, Specification, page 7, lines 12-22; Figure 1). This illustrative embodiment may be used to, for example, extend component object model (COM) objects to support persistence. This is something that Greef et al. do not teach or contemplate. Thus, for these and other reasons, claim 4 is believed to be clearly patentable over Greef et al.

Turning now to claim 5, which recites:

> 5. (Currently Amended) A method for storing and restoring user objects to persistent storage, the method comprising the steps of:
> providing one or more common interfaces that are used by the user objects to store and restore the objects to/from persistent storage [Emphasis Added];
> creating a persistence controller object for managing the persistence of the user objects, the persistence controller object being derived from at least one of the common interfaces [Emphasis Added];
> providing a plurality of user defined classes, the classes derived from a common object base class;
> creating a plurality of instances of user objects belonging to the user defined classes;
> providing a stream-in method and a stream-out method for each of the user defined classes;
> registering each added user defined class and added user object in a registry;
> grouping the objects according to class;
> storing the grouped user objects to persistent storage using the stream-out methods;
> loading the stored objects from storage into memory using the stream-in methods; and
> registering the user objects in the registry.

As can be seen, claim 5 recites, among other things, the steps of: providing one or more common interfaces that are used by the user objects to store and restore the objects to/from persistent storage; and creating a persistence controller object for managing the persistence of the user

objects, <u>the persistence controller object being derived from at least one of the common interfaces</u>. Thus, for the same reasons discussed above with respect to claim 4, as well as other reasons, claim 5 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claim 6 is also believed to be clearly patentable over Greef et al.

In addition, claim 6 recites the step of: obtaining a new address of each loaded user object, and using the stored unique identifier associated with each pointer along with the new address <u>to set each pointer value to the new address</u> [Emphasis Added]. As noted above, Greef et al. appear to only use smart pointers, and do not use conventional pointers. As such, there would be little need for Greef et al. to "set each pointer value to the new address", as recited in claim 6. For these additional reasons, dependent claim 6 is believed to be clearly patentable over Greef et al.

Turning now to claim 7, which recites:

> 7. (Currently Amended) An object adapted for persistent storage, the object having a smart pointer, <u>wherein the smart pointer includes an address attribute for containing the address of an object</u> [Emphasis Added], and an object unique identifier attribute for containing the unique identifier of an object, wherein the object smart pointer has an assignment operation which stores the address of the object being pointed to and the unique identifier of the object being pointed to, and wherein the object includes <u>a load method for using the smart pointer unique identifier attribute to determine and load a new smart pointer address attribute after the object being pointed to is loaded from persistent storage</u> [Emphasis Added].

On page 8 of the Office Action, the Examiner states that the smart pointer of Greef et al. includes an address attribute for containing the address of an object, citing column 4, lines 52-54 of Greef et al. Column 4, lines 52-57 of Greef et al. states:

> "[w]hen an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. This is done by invoking

an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier".

The Examiner concludes from this that the smart pointers of Greef et al. must inherently [Emphasis Added] contain an address of the object that it points to.

Applicant must respectfully disagree. Greef et al. also state:

> The Entity Cache comprises collections of "smart pointers" that have been created during a session. FIG. 1 shows the Entity Cache comprising a collection of new entities, a collection of dirty entities and a collection of retrieved entities. Each entity in the system has a unique object identifier and a class identifier as shown in the Entity Class depicted in FIG. 1 [Emphasis Added]. The unique object identifier is generated by the Entity ID Generator shown in FIG. 1. When the object is referenced with the entity identifier, the entity cache swivels the identifier to a smart pointer [Emphasis Added]. If a smart pointer already exists in the entity cache, it returns the smart pointer. If the smart pointer does not exist in the entity cache, the object is loaded from the nonvolatile memory and the smart pointer is returned to the entity in the entity cache. All persistent objects have an entity identifier.
>
> All entity object management is done with smart pointers. "Smart Pointers" are used to handle large amounts of objects in limited memory resources. What is required is a light weight representation of an entity, as described as light weight objects in Gamma et. al. (1995). Each entity has a corresponding smart pointer. It is a sub for a real object in memory. It it has no data members and it overloads the pointer and de-reference operators. All objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added].

(Greef et al., column 4, lines 17-41). First, and as noted above, Greef et al. state that "[a]ll objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added]." (Greef et al., column 4, lines 49-41). This suggests that the smart pointers of Greef et al. do not include both an object unique identifier attribute for containing the unique identifier of an object AND an address attribute for containing the address of an object, as recited in claim 7. Instead, the smart pointers of Greef et al. appear to only include a unique identifier, which is generated by the Entity Class depicted in Figure 1 (Greef et

al., column 4, lines 23-24).

Greef et al. also state that when the object is referenced with the entity identifier, the entity cache swivels the identifier to a smart pointer. This also indicates that the smart pointers of Greef et al. only include a unique identifier, and that the entity cache is used to swivel the identifier to a smart pointer.

The Examiner specifically points to column 4, lines 52-57 of Greef et al. However, this passage states that when an operation is invoked on a smart pointer, the object that it points to is faulted into the entity cache if it does not already exist. However, this passage also states that this is done by invoking an operation on the Data Store that can find an object in the nonvolatile memory when it is provided with the object's unique identifier. Thus, in this example, the smart pointer appears to store and deliver an object's unique identifier to the Data Store, and it is the Data Store, and not the smart pointer, that finds the object in the non-volatile memory.

In view of the foregoing, Applicant does not believe it can be argued that the smart pointers of Greef et al. inherently [Emphasis Added] include both an object unique identifier attribute for containing the unique identifier of an object AND an address attribute for containing the address of an object, as recited in claim 7. As noted in MPEP § 2112(IV), the fact that a certain result or characteristic may occur or be present in the prior art is not sufficient to establish the inherency of that result or characteristic. To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present [Emphasis Added] in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency may not be established by probabilities or possibilities.

In addition to the foregoing, Greef et al. do not appear to suggest an object that includes a

load method for using the smart pointer unique identifier attribute to determine and load a new smart pointer address attribute [Emphasis Added] after the object being pointed to is loaded from persistent storage, as recited in claim 7. As noted above, it does not appear that the smart pointers of Greef et al. even have an address attribute. In addition, Greef et al. state that it is the Data Store that finds an object in the nonvolatile memory when it is provided with the object's unique identifier, and not a load method of the object itself.

For these and other reasons, claim 7 is believed to be clearly patentable over Greef et al. For similar and other reasons, dependent claim 8 is also believed to be clearly patentable over Greef et al. Claim 9 has been canceled without prejudice.

Turning now to claim 10, which recites:

> 10. (Currently Amended) A method for writing computer objects to persistent storage, the method including the steps of:
>
> providing a plurality of software objects to be stored, wherein the objects are instantiations of at least one class to be storable, wherein the storage is in persistent storage, wherein each of the classes has a unique class ID, and wherein each of the objects has a unique object ID;
>
> providing smart pointers for at least some objects, wherein the smart pointers include an address portion to contain the address of the object being pointed to and an object identifier portion to contain an object identifier of the object being pointed to [Emphasis Added];
>
> providing a persistent object controller for controlling the lifecycle of objects to be saved to persistent storage and loaded from persistent storage;
>
> providing a Persistent Object Registry for maintaining a database of objects to be saved to persistent storage, wherein the Persistent Object Registry is in communication with the persistent controller object;
>
> providing a first save method to save all objects in the Persistent Object Registry to persistent storage;
>
> providing a second save method for saving the attributes of each class having objects to be saved to persistent storage, wherein the second save method is called by the first save method;
>
> providing a first load method for loading all objects saved in a file in persistent storage;

providing a second load method for loading the attributes of each class having objects to be loaded from persistent storage, wherein the second load method is called by the first load method;

registering the objects to be saved with the Persistent Object Registry using the persistent object controller, including storing the class ID and object ID of the objects to be saved;

writing the objects to be saved to persistent storage using the first save method and second save method;

reading the objects stored from persistent storage using the first load method and second load method;

registering the objects loaded into the Persistent Object Registry; and

resolving the smart pointer object address attributes by <u>using the object ID attribute value to search the Persistent Object Registry to retrieve the current address of the object being pointed to</u> [Emphasis Added].

As noted above, Greef et al. do not disclose or suggest smart pointers that include an address portion to contain the address of the object being pointed to <u>AND</u> an object identifier portion to contain an object identifier of the object being pointed to. In addition, Greef et al. do not disclose or suggest the step of resolving the smart pointer object address attributes by <u>using the object ID attribute value to search the Persistent Object Registry to retrieve the current address of the object being pointed to</u> [Emphasis Added].

In addition, Greef et al. do not appear to disclose or suggest many of the other elements of claim 10. For example, Greef et al. do not appear to suggest providing <u>a first save method</u> [Emphasis Added] to save all objects in the Persistent Object Registry to persistent storage <u>and a second save method</u> [Emphasis Added] for saving the attributes of each class having objects to be saved to persistent storage, <u>wherein the second save method is called by the first save method</u> [Emphasis Added]. On page 10 of the Office Action, the Examiner cites to item 101 of Figure 7 of Greef et al. as corresponding to the first save method, and item 102 of Figure 7 as corresponding to the second save method. With respect to items 101 and 102, Greef et al. state:

> A new DataCursor object and the object identity for the object to be stored are passed as parameters of this method 101. The data store then invokes the Storer method on the object to be stored 102. The next step 103 (designated by the letter "A") has the object (which is an entity) fill the DataCursor's buffer with class member data, class identifiers and class data offsets. Once we have a buffer of information representing the record of the object as shown in FIG. 6, the particular implementation of the data store saves it to permanent storage using the object identifier as the unique key 104. The process is then stopped 105.

(Greef et al., column 5, lines 53-64). As can be seen, item 101 represents the step of passing a new DataCursor object and the object identity for the object to be stored. This is clearly not equivalent to "providing a first save method to save all <u>objects</u> [Emphasis Added] in the Persistent Object Registry to persistent storage", as recited in claim 10. Item 102 of Greef et al. represents the step of invoking the Storer method on the object to be stored. This is clearly not equivalent to providing <u>a second save method</u> [Emphasis Added] for saving the attributes of each <u>class</u> having objects to be saved to persistent storage. Finally, there is no indication that the first save method calls the second save method.

Likewise, Greef et al. do not appear to suggest providing a first load method for loading all <u>objects</u> [Emphasis Added] saved in a file in persistent storage, and providing a second load method for loading the attributes of each <u>class</u> [Emphasis Added] having objects to be loaded from persistent storage, wherein the second load method is called by the first load method. On page 10 of the Office Action, the Examiner cites to item 301 of Figure 9 of Greef et al. as corresponding to the first load method and item 302 of Figure 9 as corresponding to the second load method. With respect to Figure 9, Greef et al. state:

> Retrieving objects from the permanent storage system shown in Flowchart FIG. 9 depicts the retrieving of objects that are stored in the nonvolatile storage. FIG. 9 at 300 starts the process, when we have the identification of the object to be retrieved from non-volatile store. The DataStore for a particular permanent

> storage implementation is invoked with the object identifier and the data cursor. This in turn invokes the implementation dependent persistent manager's fetch method 302 and the block of data related to the object identifier key is retrieved 303 from the permanent store in a data cursor buffer. The step designated by the letter "B" is then performed where the object is created in the volatile store 304. The retrieval method is then completed and the process stops at 305.

From this, Applicant does not believe it can readily be argued that Greef et al. discloses providing a first load method for loading all <u>objects</u> [Emphasis Added] saved in a file in persistent storage, and providing a second load method for loading the attributes of each <u>class</u> [Emphasis Added] having objects to be loaded from persistent storage, wherein the second load method is called by the first load method, as recited in claim 10.

For the reasons given above, as well as other reasons, claim 10 is believed to be clearly patentable over Greef et al. For similar and other reasons, claim 11 is also believed to be clearly patentable over Greef et al.

Turning now to claim 12, which recites:

> 12. (Previously Presented) A method for managing persistent object lifecycles, the method comprising the steps of:
>
> providing for each object a unique object identifier attribute, an object type attribute, <u>and an object address</u> [Emphasis Added];
>
> providing an object registry object for maintaining a correspondence between said unique object identifier attribute, said object address, and said object type attributes;
>
> creating a first object having a first object type, a first object address, and a first unique object identifier, and storing said first unique object identifier, address, and type in said object registry;
>
> creating a second object having a second object type, a second object address, and a second unique object identifier, and storing said second unique object identifier, address, and type in said object registry, said second object having a pointer attribute set equal to said first object address;
>
> providing said second object pointer attribute to said object registry and obtaining said first object unique identifier corresponding to said second object pointer attribute in return;

> writing said second object to persistent storage as second object data, and writing said first object unique identifier corresponding to said second object pointer attribute to persistent storage, such that said written first object unique identifier is associated with said second object pointer attribute in persistent storage;
>
> deleting said second object from non-persistent storage;
>
> reading said second object data from persistent storage and creating said second object having said second object type;
>
> reading said first object unique identifier associated with said second object data from persistent storage;
>
> providing said object registry with said first object unique identifier and obtaining said first object address in return; and
>
> setting said second object pointer attribute equal to said first object address, such that said second object pointer attribute again points to said first object [Emphasis Added].

As noted above, Greef et al. does not disclose or suggest the step of setting said second object pointer attribute equal to said first object address [Emphasis Added], such that said second object pointer attribute again points to said first object, as recited in claim 12. Instead, Greef et al. state that "[a]ll objects manipulate smart pointers. Objects have lists of smart pointers, not pointers to themselves [Emphasis Added]." (Greef et al., column 4, lines 49-41). Halter does not appear to add anything to Greef et al. in this regard. Thus, for these and other reasons, claim 12 is believed to be clearly patentable over Greef et al. in view of Halter.

Finally, Applicant has added newly presented claims 13-24, which are all believed to be clearly patentable over the prior art of record.

In view of the foregoing, Applicant believes that all pending claims 1-8, 10-24 are in condition for allowance. Reexamination and reconsideration are respectfully requested. If the Examiner believes it would be beneficial to discuss the application or its examination in any way, please call the undersigned attorney at (612) 359-9348.

Respectfully submitted,

John Quernemoen

By his attorney,

Dated: November 16, 2004

Brian N. Tufte, Reg. No. 38,638
CROMPTON, SEAGER & TUFTE, LLC
1221 Nicollet Avenue, Suite 800
Minneapolis, MN 55403-2402
Telephone: (612) 677-9050
Facsimile: (612) 359-9349